
EMBEDDED **SOLUTIONS**

RC1000-II

Software User Guide

Xilinx, XBLOX and XACTStep are trademarks of Xilinx Corp.
Microsoft and MS-DOS are registered trademarks and Windows, Windows 95 and Windows NT are trademarks of Microsoft Corporation.

This manual was written by Matthew Bowen.

© Embedded Solutions Limited. All rights reserved
Version 1.0

Table of Contents

Conventions.....	v
1. INTRODUCTION.....	1-1
1.1 About This Manual	1-2
1.2 About the Software	1-3
2. INSTALLATION	2-1
2.1 System Requirements.....	2-2
2.2 Installing the RC1000-II Support Software	2-3
2.3 Installing the Documentation Browser.....	2-4
2.3.1 Accessing the documentation	2-4
2.4 Directory Structure	2-5
3. EXAMPLE PROGRAMS	3-1
3.1 Introduction.....	3-2
3.2 Checking the RC1000-II Hardware.....	3-3
3.3 The addone Example Program	3-4
3.4 The fib Example Program	3-5
3.5 The calc Example Program.....	3-6
3.6 Compiling the Simple Example Programs	3-7
3.6.1 Compiling the Host Example Programs	3-7
3.6.2 Compiling the FPGA Example Programs.....	3-7
3.6.3 Converting the FPGA Configuration Files.....	3-8
3.7 The video Example Program	3-9
3.8 Compiling the video example program	3-10
3.8.1 Compiling the Host Program	3-10
3.8.2 Compiling the FPGA Programs	3-10
3.8.3 Converting the FPGA Configuration Files.....	3-11
4. USING THE SUPPORT SOFTWARE	4-1
4.1 Introduction.....	4-2
4.2 Writing Host Programs	4-3

4.2.1	Using the host interface library.....	4-3
4.2.2	Initialising the hardware and software.....	4-4
4.2.3	Setting the programmable clock rate.....	4-4
4.2.4	Configuring the FPGA.....	4-5
4.2.5	Communicating with the FPGA.....	4-7
4.2.6	Starting up.....	4-8
4.2.7	Cleaning up.....	4-8
4.3	Writing FPGA Programs in Handel-C.....	4-9
4.3.1	Initialising the Handel-C support software.....	4-9
4.3.2	Communicating with the host.....	4-10
4.3.3	Communicating with the daughter module.....	4-11
5.	UTILITIES.....	5-1
5.1	Introduction.....	5-2
5.2	The gencfg Utility.....	5-3
5.3	The loadfpga Utility.....	5-5

Conventions

A number of conventions are used in this document. These conventions are detailed below.



Warning Message. These messages warn you that actions may damage your hardware.



Handy Note. These messages draw your attention to crucial pieces of information.

Hexadecimal numbers appear in this document. They are prefixed with '0x' (in common with standard C syntax).

Sections of code or commands that you must type are given in typewriter font like this:

```
void main();
```

Information about a type of object you must specify is given in italics like this:

```
copy SourceFileName DestinationFileName
```

1. Introduction

1.1 About This Manual

This manual provides details of the installation and use of the support software for the Embedded Solutions RC1000-II reconfigurable computer platform. Details of the support functions are given in the RC1000-II Function Reference Manual and details of the hardware are given in the RC1000-II Hardware Reference Manual and extensive references are made throughout to these companion documents.

This chapter gives an overview of the RC1000-II system.

Chapter 2 details how to install the software support on your PC. For details of installing the hardware, consult the RC1000-II Hardware Reference Manual.

Chapter 3 details the example programs supplied with the RC1000-II package. These examples are designed to show how to write host programs that talk to the RC1000-II hardware and how to write FPGA programs that talk to the host PC. The FPGA example programs are written using the Handel-C language.

Chapter 4 describes the support offered by the RC1000-II software package in more detail. This support includes host PC libraries and FPGA macros.

Chapter 5 describes the host utilities supplied with the RC1000-II package. These utilities provide easy handling of FPGA configuration files and enable the RC1000-II FPGA to be configured without writing a host support program.

1.2 About the Software

The RC1000-II Support Software provides host libraries to simplify the process of initialising and talking to the hardware. A number of example programs and utilities are also provided to serve as a starting point for the development of your own applications.

The software provides a number of groups of host functions:

- Initialisation functions
- Functions to handle FPGA configuration files
- Functions to control the RC1000-II programmable clock
- Functions to transfer data to and from the RC1000-II FPGA
- Functions to help with error checking and debugging

The host software comes in the form of a static C library which can be linked to host programs and a set of Handel-C macros for use with FPGA programs.

2. Installation

2.1 System Requirements

The Handel-C compiler requires the following platform:

- IBM PC compatible computer.
- Windows 95 operating system.
- Microsoft Visual C++ version 4.0 or later.
- 8Mb RAM.
- Up to 2Mb hard disk space.
- CD ROM drive.

You will also need tools to target the FPGA. Embedded Solutions Ltd recommend using the Handel-C compiler and Xilinx XACTStep software version 5.2.1 or later (including M1 versions). Contact Embedded Solutions Ltd for further details of these products.

(If you are using an earlier version of the XACTStep software, please contact Embedded Solutions Ltd. for version compatibility.)

2.2 Installing the RC1000-II Support Software

This section details how to install the RC1000-II compiler.

To install for Windows 95:

1. Insert the RC1000-II Support Software CD in the CD-ROM drive.
2. Open an MS-DOS window.
3. From the DOS prompt, run the **setup** program in the root directory of the CD. E.g., if your CD ROM drive is drive **d:** type:

d:\setup

4. Follow the on screen instructions.

By default, the support software is installed in the directory **Embedded Solutions Ltd\RC1000-II\Win95**. All directory paths given in this manual start from the **RC1000-II** directory.

When you install the RC1000-II Support Software, you are given the option of a typical, compact, or custom installation. The compact installation does not install the examples and documentation. The custom installation allows you to choose whether to install them or not. The exact disk usage statistics can be viewed from the RC1000-II Support Software Package installation program. If you do not install the documentation you will still be able to read the documents from the CD.

You need the Adobe Acrobat Reader software to read and display the documents. The next section describes how to install this software.

2.3 Installing the Documentation Browser

The Handel-C documentation is provided in a format suitable for the Adobe Acrobat Reader software. If you do not already have this software installed, you should follow these instructions to install it.

1. Insert the RC1000-II Support Software CD in your CD-ROM drive.
2. Open an MS-DOS window.
3. From the prompt run the **ar32e30** program in the root directory of the CD. For example, if your CD ROM is drive **d:** then type:

d:\ar32e30

4. Follow the on screen instructions

For the latest version download from the <http://www.adobe.com> web site.

2.3.1 Accessing the documentation

The documentation is in three files:

SWref.pdf	Software User Guide (this document).
Fnref.pdf	Software Function Reference Manual.
HWref.pdf	Hardware Reference Manual

If you chose to install the documentation on your hard disk when you installed the RC1000-II Support Software, the documentation files will be stored in the **doc** subdirectory.

The documentation files are also stored in the root directory of the RC1000-II Support Software CD-ROM.

2.4 Directory Structure

Upon successful installation, the following directories and files should have been created on your hard disk. If you chose not to perform a typical installation, some of these files may be missing.

README.TXT	Text file containing latest information.
examples	
➤ addone.exe	Simple example to add one to a number.
➤ addone.bit	FPGA configuration file for the addone example program.
➤ calc.exe	Simple calculator example.
➤ examples.mak	Microsoft Visual C++ project file for simple example programs.
➤ fib.exe	Simple example to generate Fibonacci numbers.
➤ video.exe	More advanced example program to demonstrate real time video processing.
➤ video.mak	Microsoft Visual C++ project file for video example program.
doc	
➤ Fnref.pdf	RC1000-II Function Reference Manual.
➤ HWref.pdf	RC1000-II Hardware Reference Manual.
➤ SWref.pdf	RC1000-II Software User Guide (this document).
fpga	
➤ ii1000.h	Handel-C header files for RC1000-II
➤ ii1000ip.h	FPGA support software.
include	
➤ ii1000.h	Conventional C header file for RC1000-II host support software
lib	
➤ ii1000.lib	Release version of the RC1000-II support software host library.
➤ ii1000ck.lib	Debug version of the RC1000-II support software host library.
utils	
➤ gencfg.exe	FPGA configuration to C header file conversion utility.
➤ loadfpga.exe	Host RC1000-II configuration utility.

3. Example Programs

3.1 Introduction

This chapter details the example programs provided with the RC1000-II Support Software Package. The purpose of the example programs is twofold. Firstly, they provide a quick means of testing that the RC1000-II hardware is installed and functioning correctly. Secondly, they provide a starting point for your own programs.

Four example programs are provided. These are:

- A simple program that prompts for input of a number and uses the FPGA to calculate the value of the number plus one.
- A simple program that uses the FPGA to generate members of the Fibonacci number sequence.
- A program that demonstrates reconfiguration of the FPGA to implement a simple calculator.
- A more advanced program that demonstrates using the FPGA to perform real-time video processing.

3.2 Checking the RC1000-II Hardware

After installation of your RC1000-II hardware following the instructions in the RC1000-II Hardware Reference Manual and installation of the support software as described in the previous chapter, it is recommended that you run a simple test program provided to ensure that your hardware is functioning correctly.

The test program simply configures the FPGA and flashes the LED on the board. To run the test, open a DOS prompt, change to the **utils** subdirectory and type the following:

```
loadfpga -b BaseAddress -c 10000000 flash.bit
```

Here, you should replace **BaseAddress** with the address of the board. Refer to the RC1000-II Hardware Reference Manual for further details of setting the base address of your board. For example, if your board is at address 0x300, you should type:

```
loadfpga -b 0x300 -c 10000000 flash.bit
```

This test can also be used to check the programmable clock on the board. By changing the value of the clock frequency in Hz (with the **-c** option), you should see the rate of the flashing increase or decrease. For example, to half the rate of flashing, type:

```
loadfpga -b 0x300 -c 5000000 flash.bit
```

Should you be unable to get this test program to run, you should check the installation of both the hardware and software. If it still does not work correctly, contact Embedded Solutions Ltd by email at:

support@embedded-solutions.ltd.uk

3.3 The **addone** Example Program

The **addone** example program uses the FPGA to add one to numbers typed in by the user. This example illustrates setting up the board and software, configuring the FPGA from a file, setting the programmable clock and using bi-directional communications.

To run the program, open a DOS box, move to the **examples** directory and type the following:

addone

The program will repeatedly ask you for a number and then return the number plus one back to you. To stop the program, enter zero as the number.

The **addone.c** host program performs the following actions:

1. Open and initialise the RC1000-II card.
2. Set the programmable clock to run at 15MHz.
3. Configure the FPGA from the **addone.bit** file.
4. Repeatedly write numbers to and read results back from the FPGA.

The **addone.c** Handel-C program performs the following actions:

1. Initialise the RC1000-II support software.
2. Repeatedly read numbers from and return results to the host program.

Note that the host program assumes that the RC1000-II board is set at a base address of 0x300 and that the part fitted is a Xilinx 4010E FPGA. If you have your board at a different base address, you will have to modify the **BASE_ADDRESS** macro definition in the **addone.c** host program. If you have a different FPGA fitted, you will have to modify the **set part = ...** line in the **addone.c** Handel-C program. You will then need to re-compile the example program. For re-compilation instructions, see section .

3.4 The fib Example Program

The **fib** example program uses the FPGA to generate members of the Fibonacci number sequence. This example illustrates setting up the board and software, configuring the FPGA from a **#included** FPGA image array, setting the programmable clock and using uni-directional communications.

To run the program, type the following at a DOS box prompt:

fib

(This assumes you are in the **examples** directory). The program will display the first members of the sequence (i.e. 1, 3, 5, 8, 13 etc.)

The **fib.c** host program performs the following actions:

1. Open and initialise the RC1000-II card.
2. Set the programmable clock to run at 15MHz.
3. Configure the FPGA from the **#included fibcfgBuffer** array.
4. Repeatedly read numbers from the FPGA and display.

The **fib.c** Handel-C program performs the following actions:

1. Initialise the RC1000-II support software.
2. Repeatedly calculate Fibonacci numbers and write to the host program.

Note that the host program assumes that the RC1000-II board is set at a base address of 0x300 and that the part fitted is a Xilinx 4010E FPGA. If you have your board at a different base address, you will have to modify the **BASE_ADDRESS** macro definition in the **fib.c** host program. If you have a different FPGA fitted, you will have to modify the **set part = ...** line in the **fib.c** Handel-C program. You will then need to re-compile the example program. For re-compilation instructions, see section 0.

3.5 The calc Example Program

The **calc** example program uses the FPGA as a simple calculator. This example illustrates setting up the board and software, configuring the FPGA from multiple linked in FPGA image arrays, setting the programmable clock and bi-directional communications.

To run the program, type the following at a DOS box prompt:

calc

(This assumes you are in the **examples** directory). The program will ask for a number, followed by the operation you wish to perform which can be one of addition, subtraction or multiplication. It will then ask for a second number and return the result of the calculation.

The **fib.c** host program performs the following actions:

1. Open and initialise the RC1000-II card.
2. Set the programmable clock to run at 10MHz.
3. Register the three FPGA images (one each for addition, subtraction and multiplication).
4. Repeatedly request numbers and operations and display results. For each calculation, the FPGA is reconfigured according to the requested operation.

The **calcadd.c**, **calcsb.c** and **calcmult.c** Handel-C programs are all very similar, differing only in the operations they perform. Each performs the following actions:

1. Initialise the RC1000-II support software.
2. Repeatedly read two numbers from the host and return the result of the calculation.

Note that the host program assumes that the RC1000-II board is set at a base address of 0x300 and that the part fitted is a Xilinx 4010E FPGA. If you have your board at a different base address, you will have to modify the **BASE_ADDRESS** macro definition in the **fib.c** host program. If you have a different FPGA fitted, you will have to modify the **set part = ...** line in the **calcadd.c**, **calcsb.c** and **calcmult.c** Handel-C programs. You will then need to re-compile the example program. For re-compilation instructions, see section 0.

3.6 Compiling the Simple Example Programs

This section details the re-compilation process for the simple example programs. These are the **addone**, **fib** and **calc** programs. For details of re-compiling the **video** example program, see section 3.8.

Each example can be split into two parts – the host program and the FPGA program. Some of the examples use multiple FPGA programs and reconfigure the FPGA. The RC1000-II support software package provides pre-compiled versions of the example programs. This section describes how to re-compile the example programs should you modify them in any way.

3.6.1 Compiling the Host Example Programs

The host programs are all written using the Microsoft Visual C++ compiler version 4.0 or later but they are written in conventional C (i.e. no C++ features have been used).

The **examples.mak** file can be used to re-compile the simple command line examples

3.6.2 Compiling the FPGA Example Programs

The FPGA programs are all written for the Handel-C language. A compiler for this language is available from Embedded Solutions Ltd. Contact Embedded Solutions Ltd for further details of the Handel-C language and compiler.

The FPGA programs can all be re-compiled with the command line:

```
handelc -x Program
```

where *Program* is the name of the Handel-C program to compile.

The Handel-C compiler will generate a Xilinx format netlist file with the suffix **.xnf** which must then be passed through the Xilinx place and route tools. Two batch files are provided to help with this stage of the process. If you have the XactStep version 5.2.x tools installed, you should type:

```
xactmake Program
```

If you have the M1 versions of the Xilinx tools installed, you should type:

```
m1make Program
```

In both cases, the program name should be specified without extensions.

3.6.3 Converting the FPGA Configuration Files

The Xilinx place and route tools will generate FPGA configuration files with a **.bit** suffix. As will be described in chapter 4, the RC1000-II host support software provides functionality enabling configuration files to be **#included** in host programs. To achieve this, the binary configuration file must be converted to a C header file which can be **#included** in the host program. All but the **addone.exe** example program use this approach and so all but the **addone.bit** configuration file must be converted in this way. (The **addone.exe** example program loads the **addone.bit** file directly.)

The RC1000-II support software package includes a utility to convert FPGA configuration files to C header files called **gencfg.exe**. Chapter 5 describes this utility in greater detail but its basic syntax is:

```
gencfg SourceFile DestinationFile
```

Here, *SourceFile* is the **.bit** file generated by the Xilinx place and route tools and *DestinationFile* is the filename of the C header file to generate. For each Handel-C program (except the **addone** example program) in the **fpga** subdirectory of the **examples\ExampleName** directory, there is a corresponding C header file in the **examples\ExampleName** directory. For example, the **fib.c** Handel-C program in the **examples\fib\fpga** directory has a corresponding C header file called **fibcfg.h** in the **examples\fib** directory. Should you re-compile any of the example Handel-C programs, you must convert the resulting bit file and re-compile the host program corresponding to that example.

For example, should you modify the Handel-C Fibonacci number generator, **fib.c**, you must follow these stages (assuming you are in the **examples\fib\fpga** subdirectory):

```
handelc -x fib.c  
m1make fib  
gencfg fib.bit ../fibcfg.h
```

(This assumes you are using the M1 versions of the Xilinx tools). You must then use Microsoft Visual C++ to re-compile the **fib.exe** host program.

3.7 The video Example Program

The **video** example program uses the FPGA to perform various image processing operations. This example illustrates setting up the board and software, configuring the FPGA from multiple linked in FPGA image arrays, setting the programmable clock and bi-directional communications. It also illustrates the power of the FPGA as a processing unit when coupled with the Handel-C programming language.

To run the program, type the following at a DOS box prompt:

video

(This assumes you are in the **examples** directory). To start the program off, you must open a video file from the File ➔ Open menu. A video file is provided on the RC1000-II support software CD in the **examples\video** directory called **video.avi**.

When the video starts playing, the original images are shown in the left half of the window with the results of the processing shown in the right half of the window. You can select different operations from the 'Operation' menu. Five operations are provided:

1. Image thresholder
2. 3x3 Sobel edge detector
3. Red colour plane extraction
4. Mapping around a shaded sphere
5. Mapping to a constantly changing shape

Note that the host program assumes that the RC1000-II board is set at a base address of 0x300 and that the part fitted is a Xilinx 4010E FPGA. If you have your board at a different base address, you will have to modify the **BASE_ADDRESS** macro definition in the **video.h** host program. If you have a different FPGA fitted, you will have to modify the **set part = ...** line in the **thresh.c**, **edge.c**, **colour.c**, **sphere.c** and **mould.c** Handel-C programs. You will then need to re-compile the example program. For re-compilation instructions, see below.

3.8 Compiling the video example program

The **video** example can be split into two parts – the host program and the FPGA programs. Multiple FPGA programs are used – one for each of the operations. The RC1000-II support software package provides pre-compiled versions of the **video** example program. This section describes how to re-compile the **video** example program should you modify it in any way.

3.8.1 Compiling the Host Program

The host program was written for the Microsoft Visual C++ compiler version 4.0 or later using MFC.

The **demo.mak** file can be used to re-compile the **demo.exe** example program.

3.8.2 Compiling the FPGA Programs

The FPGA programs are all written using the Handel-C language. A compiler for this language is available from Embedded Solutions Ltd. Contact Embedded Solutions Ltd for further details of the Handel-C language and compiler.

The FPGA programs can all be re-compiled with the command line:

```
handelc -x Program
```

where *Program* is the name of the Handel-C program to compile.

The Handel-C compiler will generate a Xilinx format netlist file with the suffix **.xnf** which must then be passed through the Xilinx place and route tools. Two batch files are provided to help with this stage of the process. If you have the XactStep version 5.2.0 tools installed, you should type:

```
xactmake Program
```

If you have the M1 versions of the Xilinx tools installed, you should type:

```
m1make Program
```

In both cases, the program name should be specified without extensions.

3.8.3 Converting the FPGA Configuration Files

The Xilinx place and route tools will generate FPGA configuration files with a **.bit** suffix. As will be described in chapter 4, the RC1000-II host support software provides functionality enabling configuration files to be linked into host programs. To achieve this, the binary configuration file must be converted to a C header file which can be **#included** in the host program. The **video** example program uses this approach and so all the configuration files must be converted in this way.

The RC1000-II support software package includes a utility to convert FPGA configuration files to C header files called **gencfg.exe**. Chapter 5 details this utility in greater detail but its basic syntax is:

```
gencfg SourceFile DestinationFile
```

Here, *SourceFile* is the **.bit** file generated by the Xilinx place and route tools and *DestinationFile* is the filename of the C header file to generate. For each Handel-C program in the **fpga** subdirectory of the **examples\video** directory, there is a corresponding C header file in the **examples\video** directory. For example, the **thresh.c** Handel-C program in the **examples\video\fpga** directory has a corresponding C header file called **threshcfg.h** in the **examples\video** directory. Should you re-compile any of the example Handel-C programs, you must convert the resulting bit file and re-compile the host program.

For example, should you modify the thresholder, **thresh.c**, you must follow these stages (assuming you are in the **examples\video\fpga** subdirectory):

```
handelc -x thresh.c  
m1make thresh  
gencfg thresh.bit ..\threshcfg.h
```

(This assumes you are using the M1 versions of the Xilinx tools). You must then use Microsoft Visual C++ to re-compile the **video.exe** program.

4. Using the Support Software

4.1 Introduction

This chapter details the components of the RC1000-II support software library and describes step-by-step how to write your own programs for use with the RC1000-II board.

The chapter is split into two sections, one each for the host software and the FPGA software.

4.2 Writing Host Programs

Each host program must follow a number of basic steps to set up the RC1000-II hardware and prepare the board for communications. These steps are detailed below.

4.2.1 Using the host interface library

The RC1000-II support software is provided in the form of a statically linked library. This consists of a number of files:

File	Description
ii1000.h	C header file
ii1000.lib	Release version of the support library
ii1000ck.lib	Debug version of the support library

Any host program that uses the RC1000-II support software must include the **ii1000.h** header file by adding the following line to the start of the program:

```
#include "ii1000.h"
```

This assumes that the **include** directory of the RC1000-II support software is in the C compiler's include search path. The include search path in Microsoft Visual C++ V4.x can be set from the Tools ➤ Options ➤ Directories menu. For other compilers, refer to your compiler documentation for details of how to set the include search path.

The host program must also be linked with one of the two versions of the libraries. The **ii1000ck.lib** file should be used when debugging applications as it contains full checking on arguments to the functions. Since this creates a large overhead on some functions, the **ii1000.lib** file performs only essential checking on arguments and, hence, some functions execute more quickly.



Note that some functions can cause your computer to hang when used incorrectly so it is strongly recommended that you use the **ii1000ck.lib file until you are confident that your application is fully debugged.**

To link the library file with your host program, you must add one of the **.lib** files to your linker command line. To add the file using Microsoft Visual C++ V4.x, type the name of the file in the Object/library modules edit box in the Build ➤ Settings ➤ Link ➤ General dialog box. For other compilers, refer to your compiler documentation for details of how to link additional libraries.

4.2.2 Initialising the hardware and software

The first step for any program is to initialise the RC1000-II hardware and support software. This is done by calling the **II1000OpenCard()** function specifying the card's base address and FPGA part type. This function will return a handle which must be used to identify the RC1000-II board in future calls to the support software.

Refer to the RC1000-II Hardware Reference Manual for details of how to identify free base addresses and setting the base address of your RC1000-II board.

For example, to obtain a handle for a board with a Xilinx 4010 device at a base address of 0x300, you could use the following lines of code:

```
II1000_STATUS Status;  
II1000_HANDLE Handle;  
  
Status=II1000OpenCard(0x300, II1000_PART_4010, &Handle);  
if (II1000_IS_ERROR(Status))  
{  
    ReportError(Status);  
}
```

Here, the **II1000OpenCard()** function will return an error if the board could not be opened. The **II1000_IS_ERROR()** macro can be used to determine whether the return code is OK, a warning or a serious error. The **ReportError()** function must be supplied elsewhere in the program to deal with functions that fail.

4.2.3 Setting the programmable clock rate

The RC1000-II board has a programmable clock as one of the clock sources for the FPGA (the others being a fixed ISA bus clock and 8MHz and 32MHz Industry Pack clocks – see the Hardware Reference manual for how to select the clock). The RC1000-II support software provides a function to set the programmable clock period for FPGA designs that use this clock. This rate should be set before configuring the FPGA to ensure that the design is not over-clocked when it first starts.

To set the programmable clock rate, use the **II1000SetClockRate()** function specifying the handle of the board and the required rate in Hertz. For example, to set the clock to 10MHz you could use the following code:

```
Status=II1000SetClockRate(Handle, 10000000);  
if (II1000_IS_ERROR(Status))  
{  
    ReportError(Status);  
}
```

Note that if you are using one of the other clock sources, you need not set the programmable clock rate.

4.2.4 Configuring the FPGA

Once the board has been initialised and the clock set up, you must configure the FPGA with your FPGA image file. There are three ways of configuring from FPGA image files:

1. Configuring directly from a file using the **II1000ConfigureFromFile()** function.
2. Loading an FPGA image file into memory with the **II1000LoadFile()** function ready for configuration later with the **II1000ConfigureFPGA()** function.
3. Including a static array containing the configuration information in the host code. This can be registered with the support software with the **II1000RegisterImage()** function to obtain a handle for use later with the **II1000ConfigureFPGA()** function.

The first option is the simplest but requires re-reading the configuration file every time the FPGA is configured which can be a significant overhead when frequently re-configuring the FPGA. The second option overcomes this limitation while the third option allows fast configuration without the need for multiple files. The third option means that a single executable can be used which contains the host program and the FPGA configuration information.

The RC1000-II support software package contains a utility called **gencfg** to generate static arrays from FPGA configuration files to help with option 3 above. Refer to chapter 5 for further details of this utility.

All file handling routines can read configuration files in three formats:

1. Xilinx binary BIT files (**.bit** extension)
2. Xilinx raw BIT files (**.rbit** extension)
3. Motorola exormax PROM format files (**.exo** extension)

In each case, the contents of the file are analysed to determine the format rather than using the filename extension so the extension may be any set of letters.

For example, to configure the FPGA directly from a file called **config.bit** you could use the following code:

```
Status=II1000ConfigureFromFile(Handle, "config.bit");
if (II1000_IS_ERROR(Status))
{
    ReportError(Status);
}
```

To read the file into memory and then configure later you could use the following code:

```
II1000_IMAGE Image;

Status=II1000LoadImage("config.bit", &Image);
if (II1000_IS_ERROR(Status))
{
    ReportError(Status);
}

.....

Status=II1000ConfigureFPGA(Handle, Image);
if (II1000_IS_ERROR(Status))
{
    ReportError(Status);
}
```

To register a static image array and then configure later you could use the following code:

```
II1000_IMAGE Image;

Status=II1000RegisterImage(configBuffer, configLength,
                           configPart, &Image);
if (II1000_IS_ERROR(Status))
{
    ReportError(Status);
}

.....

Status=II1000ConfigureFPGA(Handle, Image);
if (II1000_IS_ERROR(Status))
{
    ReportError(Status);
}
```

4.2.5 Communicating with the FPGA

The RC1000-II support software provides four functions to allow communications between the host and FPGA. These functions are split into two pairs. The first pair is used to check whether the board is ready for the communication to proceed before transferring data and the second pair assumes the board is ready for the transaction.

The **II1000PollRead()** and **II1000PollWrite()** functions are used when you wish to check whether a board is ready for a read or write before actually performing the transfer.

The **II1000ReadData()** and **II1000WriteData()** functions are used when you know that the board is ready for the communication. If the board is not ready for the communication then the hardware will ensure that no data is lost but will halt the host processor while doing waiting for the board. Should the board never become ready then the system will hang. Therefore, it is only recommended that you use these functions when you are confident that your system is working correctly. The advantage of using these functions over the polling functions is that the transfer rate will be much higher because they only perform one access on the ISA bus whereas the polling functions require at least two accesses.

With both pairs of functions you can write 19 bits of data to the board and read 16 bits back. This is because 16-bit data transfers are used on the ISA bus, and also address bits 1-3 are written to the FPGA (but not read back) to give a total of 19 bits for writes.

For example, to read a word from the board when the board becomes ready to send data, you could use the following code:

```
unsigned short ReturnVal;  
do  
{  
    Status=II1000PollRead(Handle, &ReturnVal);  
} while (Status==II1000_STATUS_NOT_READY);
```

The value read would then be in the **ReturnVal** variable.

To write the value 23 to a board without checking that the board is ready to receive the data you could use the following code:

```
Status=II1000WriteData(Handle, 23);  
if (II1000_IS_ERROR(Status))  
{  
    ReportError(Status);  
}
```

4.2.6 Starting up

Since it is not always possible to ensure that the latch on the RC1000-II board is empty when the FPGA is first configured (either because the previous configuration left a word in the latch or because there were glitches on some of the FPGA pins during configuration), you should ensure that the latch is empty before starting any communications with the board. This can be done by using the following code immediately after configuring the FPGA:

```
II1000PollRead(Handle, &ReturnVal);
```

This will read a word of garbage from the latch if the latch is full.

4.2.7 Cleaning up

Before exiting from your host program, you should free any FPGA images you allocated with the **II1000LoadFile()** or **II1000RegisterImage()** functions by calling the **II1000FreeImage()** function for each one. For example, to free the image **Image**, you could use the following code:

```
Status=II1000FreeImage(Image);  
if (II1000_IS_ERROR(Status))  
{  
    ReportError(Status);  
}
```

You should then close the board with the following code:

```
Status=II1000CloseCard(Handle);  
if (II1000_IS_ERROR(Status))  
{  
    ReportError(Status);  
}
```

4.3 Writing FPGA Programs in Handel-C

The RC1000-II support software package contains a Handel-C include file containing a number of macros to allow simple communications to and from the host. The support program is called **ii1000.h** in the **fpga** directory. To use the support software, you must include the following line at the start of your Handel-C program:

```
#include "ii1000.h"
```

You must also inform the Handel-C compiler of the location of this header file. This can be done either by adding the option:

```
-cpp -IRC1000\Dir\fpga
```

to the Handel-C command line or by adding the same text to the **HANDELC_CPPFLAGS** environment variable. Refer to the Handel-C compiler documentation for further details of the compiler command line options.

4.3.1 Initialising the Handel-C support software

The Handel-C support software contains a macro, **II1000CommsTask()**, which must be executed in parallel with the main Handel-C program. This macro handles all communications to and from the host. For example, your Handel-C program may look something like this:

```
#include "ii1000.h"

void main(void)
{
    par
    {
        II1000CommsTask();

        {
            Main Program
        }
    }
}
```

The **II1000CommsTask()** macro should have no effect on the performance of your main program apart from using some of the logic within the FPGA.

4.3.2 Communicating with the host

The Handel-C support software contains three macros to communicate with the host. **II1000ReadData()** reads a 16 bit word from the host interface latch and stores the value in a variable. **II1000ReadAllData()** reads 19 bits of data from the host interface and stores the value in a variable. The additional 3 bits of data come from the address latch (refer to the RC1000-II Hardware Reference Manual for further details). Neither macro will return until data has been successfully read from the host.

II1000WriteData() writes 16 bits of data to the host interface latch. This function will not return until data has successfully been written into the latch.

Each of these macros will return after a minimum of 2 clock cycles. Should you call one of these macros immediately after another call to any of these macros, an extra cycle of delay will automatically be inserted to ensure there are no data bus conflicts. Thus, when continuously transferring data, the minimum time for execution of these macros is 3 clock cycles each. Knowing this, it is possible to use the additional clock cycle of delay for useful processing. For example, the following two pieces of code each take 5 clock cycles to execute:

```
{
    II1000WriteData(a); // 2 cycles
    II1000WriteData(b); // 2+1 cycles
}

{
    II1000WriteData(a); // 2 cycles
    a=a+3;               // using extra cycle
    II1000WriteData(b); // 2 cycles
}
```

Note that you should never call any two of these macros in parallel. For example, all the following are illegal and would lead to unpredictable results:

```
par
{
    II1000WriteData(a);
    II1000WriteData(b); // Illegal
}

par
{
    II1000WriteData(a);
    II1000ReadData(b); // Illegal
}
```

```
par
{
    {
        a=3;
        II1000ReadData(a);
    }
    II1000ReadData(b); // Illegal
}
```

To avoid such conflicts, it is strongly recommended that all host communications take place from a single branch of a par construct.

4.3.3 Communicating with the daughter module

The RC1000-II Handel-C support software contains some macros to illustrate various access cycles to the Industry Pack daughter module. The Industry Pack support program is called **ii1000ip.h** in the **fpga** directory. To use the Industry Pack support software, you must include the following line at the start of your Handel-C program:

```
#include "ii1000ip.h"
```

You must also inform the Handel-C compiler of the location of this header file. This can be done either by adding the option:

```
-cpp -IRC1000\Dir\fpga
```

to the Handel-C command line or by adding the same text to the **HANDELC_CPPFLAGS** environment variable. Refer to the Handel-C compiler documentation for further details of the compiler command line options.

Only memory byte and word Industry Pack access cycles are provided but these illustrate how you can write macros to perform IO, ID and IntAck cycles. The macros are:

- **II1000IPWriteMemByte()** to write one byte to the daughter module with a memory cycle
- **II1000IPWriteMemWord()** to write one 16 bit word to the daughter module with a memory cycle
- **II1000IPReadMemByte()** to read one byte from the daughter module with a memory cycle
- **II1000IPReadMemWord()** to read one 16 bit word from the daughter module with a memory cycle

Each macro takes a 23 bit address and performs the standard multiplexed address/data access.

5. Utilities

5.1 Introduction

The RC1000-II support software package contains two utility programs to aid with software development. The **gencfg** utility converts FPGA image files into a C header file to allow you to link your FPGA images into your application. The **loadfpga** utility will download an FPGA image to a card and set it running. This is useful if you wish to write an FPGA program without writing a host program to go with it. For example, an FPGA program that simply talks to the daughter module without communicating to the host PC.

This chapter details the use of each of these utilities.

5.2 The gencfg Utility

As described in chapter 4, the RC1000-II support software provides functions to allow FPGA images to be included in a host executable file to avoid having many FPGA image files for one application. The **gencfg** utility takes an FPGA image file as its input and generates a C header file as its output. For example:

```
gencfg fib.bit fibcfg.h
```

This will read the FPGA configuration file **fib.bit** and generate the C header file **fibcfg.h**. The input file can be in any one of three formats:

1. Xilinx binary BIT files (**.bit** extension)
2. Xilinx raw BIT files (**.rbit** extension)
3. Motorola exormax PROM format files (**.exo** extension)

The output file will have the following general format:

```
/*
 * Following lines generated by gencfg utility from fib.bit
 */
static II1000_PART_TYPE fibcfgPart = II1000_PART_4010;
static unsigned long fibcfgLength = 0x000056fc;
static unsigned char fibcfgBuffer[] = {
0xff, 0x20, 0x2b, 0x7d, 0x9f, 0x57, 0xfe, 0xfe,
0xbf, 0xaf, 0xeb, 0xfa, 0xfe, 0xbf, 0xaf, 0xeb,
0xfa, 0xfe, 0xbf, 0xb7, 0xf5, 0xfd, 0x7f, 0x5f,
.....
```

These three definitions provide all the information required by the **II1000RegisterImage()** function. For example, to include this image in your host program, you should:

1. Include the image definitions in your host program with the line:

```
#include "fibcfg.h"
```

2. Register the image with the host support software with the line:

```
II1000RegisterImage(fibcfgBuffer,
                    fibcfgLength,
                    fibcfgPart,
                    &Image);
```

3. Configure the FPGA with the line:

```
II1000ConfigureFPGA(Handle, Image);
```

The **gencfg** utility will always name the three variables based on the destination filename. For example, the results of the command:

```
gencfg in.bit ../../dir1\fpga\dir2\out.h
```

would be:

```
/*  
 * Following lines generated by gencfg utility from in.bit  
 */  
static II1000_PART_TYPE outPart = II1000_PART_4010;  
static unsigned long outLength = 0x000056fc;  
static unsigned char outBuffer[] = {  
0xff, 0x20, 0x2b, 0x7d, 0x9f, 0x57, 0xfe, 0xfe,  
0xbf, 0xaf, 0xeb, 0xfa, 0xfe, 0xbf, 0xaf, 0xeb,  
0xfa, 0xfe, 0xbf, 0xb7, 0xf5, 0xfd, 0x7f, 0x5f,  
.....
```

The **gencfg** utility will automatically detect the input file type from its contents and automatically detect the FPGA part type from the FPGA image.

5.3 The loadfpga Utility

The **loadfpga** utility can be used to configure the RC1000-II FPGA from the command prompt without having to write a host application. The general format of the command line is:

loadfpga -b *BaseAddress* {-c *ClockRate*} {-p *Part*} *FileName*

Here, *BaseAddress* is the base address of the board which can be 0x100, 0x180, 0x200 or 0x300. Refer to the RC1000-II Hardware Reference Manual for details of setting the base address of the board.

ClockRate specifies the rate to set the programmable clock to before configuring the FPGA. This is optional for designs that do not use the programmable clock.

Part optionally specifies the part number of the FPGA fitted to the RC1000-II board. Valid values are 4003, 4005, 4006, 4008 and 4010. If this option is not specified, the **loadfpga** utility will use a part of **II1000_PART_UNKNOWN** when it opens the board. Refer to the RC1000-II Function Reference Manual for details of such a part type.

Filename specifies the FPGA image file to load. This can be in any of the following formats:

1. Xilinx binary BIT files (**.bit** extension)
2. Xilinx raw BIT files (**.rbt** extension)
3. Motorola exormax PROM format files (**.exo** extension)

Refer to the Xilinx FPGA software tools documentation for details of how to generate files of these types.

